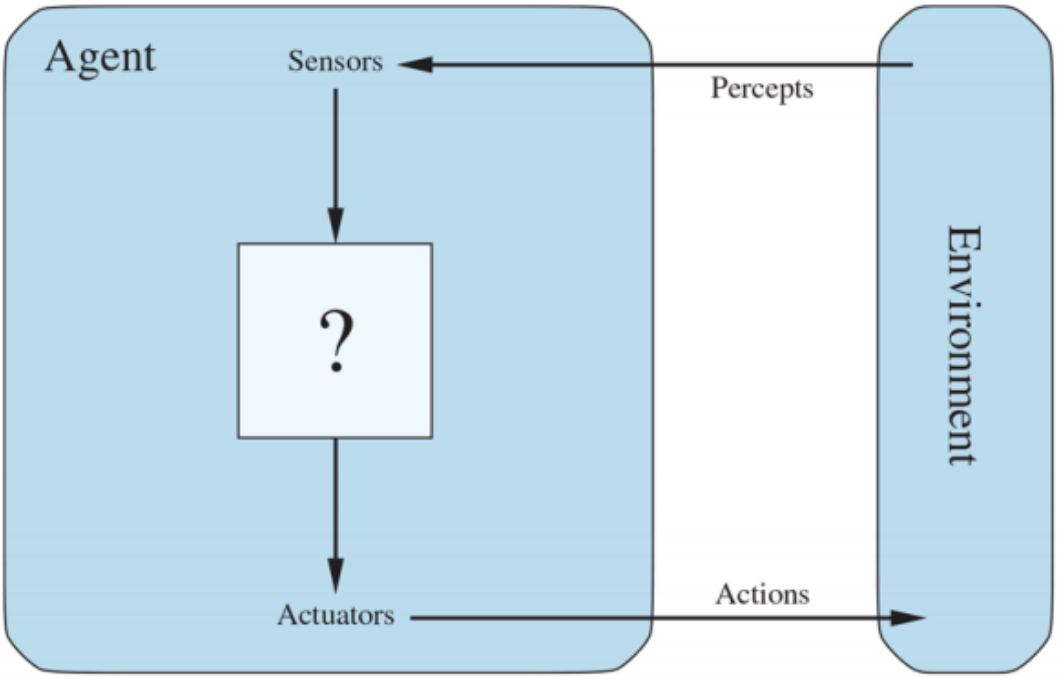
2.1 Agents and Environments

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. This simple idea is illustrated in Figure 2.1 . A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators. A software agent receives file contents, network packets, and human input (keyboard/mouse/touchscreen/voice) as sensory inputs and acts on the environment by writing files, sending network packets, and displaying information or generating sounds. The environment could be everything—the entire universe! In practice it is just that part of the universe whose state we care about when designing this agent—the part that affects what the agent perceives and that is affected by the agent's actions.

Figure 2.1



Agents interact with environments through sensors and actuators.

Figure 2.1 Agents interact with environments through sensors and actuators.

Environment

Sensor

Actuator

We use the term percept to refer to the content an agent's sensors are perceiving. An agent's percept sequence is the complete history of everything the agent has ever perceived. In general, an agent's choice of action at any given instant can depend on its built-in knowledge and on the entire

percept sequence observed to date, but not on anything it hasn't perceived. By specifying the agent's choice of action for every possible percept sequence, we have said more or less everything there is to say about the agent. Mathematically speaking, we say that an agent's behavior is described by the agent function that maps any given percept sequence to an action.

Percept

Percept sequence

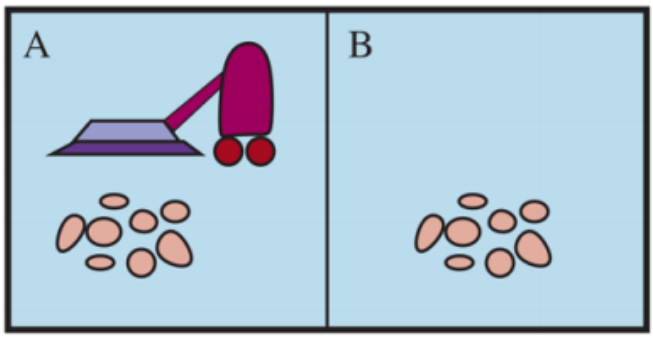
Agent function

We can imagine tabulating the agent function that describes any given agent; for most agents, this would be a very large table—infinite, in fact, unless we place a bound on the length of percept sequences we want to consider. Given an agent to experiment with, we can, in principle, construct this table by trying out all possible percept sequences and recording which actions the agent does in response. The table is, of course, an external characterization of the agent. Internally, the agent function for an artificial agent will be implemented by an agent program. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.

1 If the agent uses some randomization to choose its actions, then we would have to try each sequence many times to identify the probability of each action. One might imagine that acting randomly is rather silly, but we show later in this chapter that it can be very intelligent.

Agent program

To illustrate these ideas, we use a simple example—the vacuum-cleaner world, which consists of a robotic vacuum-cleaning agent in a world consisting of squares that can be either dirty or clean. Figure 2.2 shows a configuration with just two squares, and . The vacuum agent perceives which square it is in and whether there is dirt in the square. The agent starts in square . The available actions are to move to the right, move to the left, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square. A partial tabulation of this agent function is shown in Figure 2.3 and an agent program that implements it appears in Figure 2.8 on page 49.



A vacuum-cleaner world with just two locations. Each location can be clean or dirty, and the agent can move left or right and can clean the square that it occupies. Different versions of the vacuum world allow for different rules about what the agent can perceive, whether its actions always succeed, and so on.

Figure 2.3

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

2 In a real robot, it would be unlikely to have an actions like “move right” and “move left.” Instead the actions would be “spin wheels forward” and “spin wheels backward.” We have chosen the actions to be easier to follow on the page, not for ease of implementation in an actual robot.

Figure 2.2 1 □ A B A 2 □ □A vacuum-cleaner world with just two locations. Each location can be clean or dirty, and the agent can move left or right and can clean the square that it occupies. Different versions of the vacuum world allow for different rules about what the agent can perceive, whether its actions always succeed, and so on.

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2 . The agent cleans the current square if it is dirty, otherwise it moves to the other square. Note that the table is of unbounded size unless there is a restriction on the length of possible percept sequences.

Looking at Figure 2.3 , we see that various vacuum-world agents can be defined simply by filling in the right-hand column in various ways. The obvious question, then, is this: What is the right way to fill out the table? In other words, what makes an agent good or bad, intelligent or stupid? We answer these questions in the next section. Before closing this section, we should emphasize that the notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization

that divides the world into agents and non-agents. One could view a hand-held calculator as an agent that chooses the □ □action of displaying “4” when given the percept sequence but such an analysis would hardly aid our understanding of the calculator. In a sense, all areas of engineering can be seen as designing artifacts that interact with the world; AI operates at (what the authors consider to be) the most interesting end of the spectrum, where the artifacts have significant computational resources and the task environment requires nontrivial decision making.

2.2 Good Behavior: The Concept of Rationality A rational agent is one that does the right thing. Obviously, doing the right thing is better than doing the wrong thing, but what does it mean to do the right thing? Rational agent 2.2.1 Performance measures Moral philosophy has developed several different notions of the “right thing,” but AI has generally stuck to one notion called consequentialism: we evaluate an agent’s behavior by its consequences. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well. This notion of desirability is captured by a performance measure that evaluates any given sequence of environment states. Consequentialism Performance measure Humans have desires and preferences of their own, so the notion of rationality as applied to humans has to do with their success in choosing actions that produce sequences of environment states that are desirable from their point of view. Machines, on the other hand, do not have desires and preferences of their own; the performance measure is, initially at least, in the mind of the designer of the machine, or in the mind of the users the machine isdesigned for. We will see that some agent designs have an explicit representation of (a version of) the performance measure, while in other designs the performance measure is entirely implicit—the agent may do the right thing, but it doesn’t know why. Recalling Norbert Wiener’s warning to ensure that “the purpose put into the machine is the purpose which we really desire” (page 33), notice that it can be quite hard to formulate a performance measure correctly. Consider, for example, the vacuum-cleaner agent from the preceding section. We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift. With a rational agent, of course, what you ask for is what you get. A rational agent can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor, then cleaning it up again, and so on. A more suitable performance measure would reward the agent for having a clean floor. For example, one point could be awarded for each clean square at each time step (perhaps with a penalty for electricity consumed and noise generated). As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave. Even when the obvious pitfalls are avoided, some knotty problems remain. For example, the notion of “clean floor” in the preceding paragraph is based on average cleanliness over time. Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time while the other cleans energetically but takes long breaks. Which is preferable might seem to be a fine point of janitorial science, but in fact it is a deep philosophical question with far-reaching implications. Which is better—a reckless life of highs and lows, or a safe but humdrum existence? Which is better—an economy where everyone lives in moderate poverty, or one in which some live in plenty while others are very poor? We leave these questions as an exercise for the diligent reader. For most of the book, we will assume that the performance measure can be specified correctly. For the reasons given above, however, we must accept the possibility that we might put the wrong purpose into the machine—precisely the King Midas problem described on page 33. Moreover, when designing one piece of software, copies of which will belong to different users, we cannot anticipate the exact preferences of each individual user. Thus, we may need to

build agents that reflect initial uncertainty about the true performance measure and learn more about it as time goes by; such agents are described in Chapters 16 , 18 , and 22 . 2.2.2 Rationality What is rational at any given time depends on four things: The performance measure that defines the criterion of success. The agent's prior knowledge of the environment. The actions that the agent can perform. The agent's percept sequence to date. This leads to a definition of a rational agent: For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has. Definition of a rational agent Consider the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square if not; this is the agent function tabulated in Figure 2.3 . Is this a rational agent? That depends! First, we need to say what the performance measure is, what is known about the environment, and what sensors and actuators the agent has. Let us assume the following: The performance measure awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps. The "geography" of the environment is known a priori (Figure 2.2) but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The and actions move the agent one square except when this would take the agent outside the environment, in which case the agent remains where it is. The only available actions are , , and . The agent correctly perceives its location and whether that location contains dirt. Right Left Right Left SuckUnder these circumstances the agent is indeed rational; its expected performance is at least as good as any other agent's. One can see easily that the same agent would be irrational under different circumstances. For example, once all the dirt is cleaned up, the agent will oscillate needlessly back and forth; if the performance measure includes a penalty of one point for each movement, the agent will fare poorly. A better agent for this case would do nothing once it is sure that all the squares are clean. If clean squares can become dirty again, the agent should occasionally check and re-clean them if needed. If the geography of the environment is unknown, the agent will need to explore it. Exercise 2.VACR asks you to design agents for these cases. 2.2.3 Omniscience, learning, and autonomy Omniscience We need to be careful to distinguish between rationality and omniscience. An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality. Consider the following example: I am walking along the Champs Elysées one day and I see an old friend across the street. There is no traffic nearby and I'm not otherwise engaged, so, being rational, I start to cross the street. Meanwhile, at 33,000 feet, a cargo door falls off a passing airliner, and before I make it to the other side of the street I am flattened. Was I irrational to cross the street? It is unlikely that my obituary would read "Idiot attempts to cross street." 3 See N. Henderson, "New door latches urged for Boeing 747 jumbo jets," Washington Post, August 24, 1989. This example shows that rationality is not the same as perfection. Rationality maximizes expected performance, while perfection maximizes actual performance. Retreating from a requirement of perfection is not just a question of being fair to agents. The point is that if we expect an agent to do what turns out after the fact to be the best action, it will be impossible to design an agent to fulfill this specification—unless we improve the performance of crystal balls or time machines. 3Our definition of rationality does not require omniscience, then, because the rational choice depends only on the percept sequence to date. We must also ensure that we haven't inadvertently allowed the agent to engage in decidedly underintelligent activities. For example, if an agent does not look both ways before crossing a busy road, then its percept sequence will not tell it that there is a large truck approaching at high speed. Does our definition of rationality say that it's now OK to cross the road? Far from it! First, it would not be rational to cross the road given this uninformative percept sequence: the risk of accident from crossing without looking is too great. Second, a rational agent should choose the "looking" action before stepping into the street, because looking helps maximize the expected performance.

Doing actions in order to modify future percepts— sometimes called information gathering—is an important part of rationality and is covered in depth in Chapter 16 . A second example of information gathering is provided by the exploration that must be undertaken by a vacuum-cleaning agent in an initially unknown environment. Information gathering Our definition requires a rational agent not only to gather information but also to learn as much as possible from what it perceives. The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented. There are extreme cases in which the environment is completely known a priori and completely predictable. In such cases, the agent need not perceive or learn; it simply acts correctly. Learning Of course, such agents are fragile. Consider the lowly dung beetle. After digging its nest and laying its eggs, it fetches a ball of dung from a nearby heap to plug the entrance. If the ball of dung is removed from its grasp en route, the beetle continues its task and pantomimes plugging the nest with the nonexistent dung ball, never noticing that it is missing. Evolution has built an assumption into the beetle's behavior, and when it is violated, unsuccessful behavior results. Slightly more intelligent is the sphex wasp. The female sphex will dig a burrow, go out and sting a caterpillar and drag it to the burrow, enter the burrow again to check all is well, drag the caterpillar inside, and lay its eggs. The caterpillar serves as a food source when the eggs hatch. So far so good, but if an entomologist moves the caterpillar a few inches away while the sphex is doing the check, it will revert to the "drag the caterpillar" step of its plan and will continue the plan without modification, re-checking the burrow, even after dozens of caterpillar-moving interventions. The sphex is unable to learn that its innate plan is failing, and thus will not change it. To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts and learning processes, we say that the agent lacks autonomy. A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge. For example, a vacuum-cleaning agent that learns to predict where and when additional dirt will appear will do better than one that does not. Autonomy As a practical matter, one seldom requires complete autonomy from the start: when the agent has had little or no experience, it would have to act randomly unless the designer gave some assistance. Just as evolution provides animals with enough built-in reflexes to survive long enough to learn for themselves, it would be reasonable to provide an artificial intelligent agent with some initial knowledge as well as an ability to learn. After sufficient experience of its environment, the behavior of a rational agent can become effectively independent of its prior knowledge. Hence, the incorporation of learning allows one to design a single rational agent that will succeed in a vast variety of environments.

2.3 The Nature of Environments

Now that we have a definition of rationality, we are almost ready to think about building rational agents. First, however, we must think about task environments, which are essentially the "problems" to which rational agents are the "solutions." We begin by showing how to specify a task environment, illustrating the process with a number of examples. We then show that task

environments come in a variety of flavors. The nature of the task environment directly affects the appropriate design for the agent program.

Task environment 2.3.1

Specifying the task environment In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent's actuators and sensors. We group all these under the heading of the task environment. For the acronymically minded, we call this the PEAS (Performance, Environment, Actuators, Sensors) description. In designing an agent, the first step must always be to specify the task environment as fully as possible.

PEAS

The vacuum world was a simple example; let us consider a more complex problem: an automated taxi driver. Figure 2.4 summarizes the PEAS description for the taxi's task environment. We discuss each element in more detail in the following paragraphs. Figure 2.4

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

PEAS description of the task environment for an automated taxi driver.

PEAS description of the task environment for an automated taxi driver.

First, what is the performance measure to which we would like our automated driver to aspire? Desirable qualities include getting to the correct destination; minimizing fuel consumption and wear and tear; minimizing the trip time or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.

Next, what is the driving environment that the taxi will face? Any taxi driver must deal with a variety of roads, ranging from rural lanes and urban alleys to 12-lane freeways. The roads contain other

traffic, pedestrians, stray animals, road works, police cars, puddles, and potholes. The taxi must also interact with potential and actual passengers. There are also some optional choices. The taxi might need to operate in Southern California, where snow is seldom a problem, or in Alaska, where it seldom is not. It could always be driving on the right, or we might want it to be flexible enough to drive on the left when in Britain or Japan. Obviously, the more restricted the environment, the easier the design problem.

The actuators for an automated taxi include those available to a human driver: control over the engine through the accelerator and control over steering and braking. In addition, it will need output to a display screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles, politely or otherwise.

The basic sensors for the taxi will include one or more video cameras so that it can see, as well as lidar and ultrasound sensors to detect distances to other cars and obstacles. To avoid speeding tickets, the taxi should have a speedometer, and to control the vehicle properly, especially on curves, it should have an accelerometer. To determine the mechanical state ofthe vehicle, it will need the usual array of engine, fuel, and electrical system sensors. Like many human drivers, it might want to access GPS signals so that it doesn't get lost. Finally, it will need touchscreen or voice input for the passenger to request a destination.

In Figure 2.5 , we have sketched the basic PEAS elements for a number of additional agent types. Further examples appear in Exercise 2.PEAS. The examples include physical as well as virtual environments. Note that virtual task environments can be just as complex as the "real" world: for example, a software agent (or software robot or softbot) that trades on auction and reselling Web sites deals with millions of other users and billions of objects, many with real images.

Figure 2.5

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	Correct categorization of objects, terrain	Orbiting satellite, downlink, weather	Display of scene categorization	High-resolution digital camera
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, tactile and joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, raw materials, operators	Valves, pumps, heaters, stirrers, displays	Temperature, pressure, flow, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, feedback, speech	Keyboard entry, voice

Examples of agent types and their PEAS descriptions.

Figure 2.5 Examples of agent types and their PEAS descriptions.

Software agent

Softbot

2.3.2 Properties of task environments The range of task environments that might arise in AI is obviously vast. We can, however, identify a fairly small number of dimensions along which task environments can be categorized. These dimensions determine, to a large extent, the appropriate agent design and the applicability of each of the principal families of techniques for agent implementation. First we list the dimensions, then we analyze several task environments to illustrate the ideas. The definitions here are informal; later chapters provide more precise statements and examples of each kind of environment.

Fully observable

Partially observable

FULLY OBSERVABLE VS. PARTIALLY OBSERVABLE:

If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are relevant to the choice of action; relevance, in turn, depends on the performance measure. Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world. An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data—for example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and anautomated taxi cannot see what other drivers are thinking. If the agent has no sensors at all then the environment is unobservable. One might think that in such cases the agent's plight is hopeless, but, as we discuss in Chapter 4 , the agent's goals may still be achievable, sometimes with certainty.

Unobservable

SINGLE-AGENT VS. MULTIAGENT: The distinction between single-agent and multiagent environments may seem simple enough. For example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a twoagent environment. However, there are some subtle issues. First, we have described how an entity may be viewed as an agent, but we have not explained which entities must be viewed as agents. Does an agent (the taxi driver for example) have to treat an object (another vehicle) as an agent, or can it be treated merely as an object behaving according to the laws of physics, analogous to waves at the beach or leaves blowing in the wind? The key distinction is whether 's behavior is best described as maximizing a performance measure whose value depends on agent 's behavior.

Single-agent

Multiagent

For example, in chess, the opponent entity is trying to maximize its performance measure, which, by the rules of chess, minimizes agent 's performance measure. Thus, chess is a competitive multiagent environment. On the other hand, in the taxi-driving environment, avoiding collisions maximizes the performance measure of all agents, so it is a partially □ A B B A B Acooperative multiagent environment. It is also partially competitive because, for example, only one car can occupy a parking space.

Competitive

Cooperative

The agent-design problems in multiagent environments are often quite different from those in single-agent environments; for example, communication often emerges as a rational behavior in

multiagent environments; in some competitive environments, randomized behavior is rational because it avoids the pitfalls of predictability.

Deterministic vs. nondeterministic.

If the next state of the environment is completely determined by the current state and the action executed by the agent(s), then we say the environment is deterministic; otherwise, it is nondeterministic. In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment. If the environment is partially observable, however, then it could appear to be nondeterministic.

Deterministic

Nondeterministic

Most real situations are so complex that it is impossible to keep track of all the unobserved aspects; for practical purposes, they must be treated as nondeterministic. Taxi driving is clearly nondeterministic in this sense, because one can never predict the behavior of trafficexactly; moreover, one's tires may blow out unexpectedly and one's engine may seize up without warning. The vacuum world as we described it is deterministic, but variations can include nondeterministic elements such as randomly appearing dirt and an unreliable suction mechanism (Exercise 2.VFIN).

One final note: the word stochastic is used by some as a synonym for "nondeterministic," but we make a distinction between the two terms; we say that a model of the environment is stochastic if it explicitly deals with probabilities (e.g., "there's a 25% chance of rain tomorrow") and "nondeterministic" if the possibilities are listed without being quantified (e.g., "there's a chance of rain tomorrow").

Stochastic

EPISODIC VS. SEQUENTIAL:

In an episodic task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. Many classification tasks are episodic. For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions; moreover, the current decision doesn't affect whether the next part is defective. In sequential environments, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential: in both cases, short-

term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

4 The word “sequential” is also used in computer science as the antonym of “parallel.” The two meanings are largely unrelated.

Episodic

Sequential

Static

Dynamic

STATIC VS. DYNAMIC:

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn’t decided yet, that counts as deciding to do nothing. If the environment itself does not change with the passage of time but the agent’s performance score does, then we say the environment is semidynamic. Taxi driving is clearly dynamic: the other cars and the taxi itself keep moving while the driving algorithm dithers about what to do next. Chess, when played with a clock, is semidynamic. Crossword puzzles are static.

Semidynamic

DISCRETE VS. CONTINUOUS: The discrete/continuous distinction applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agent. For example, the chess environment has a finite number of distinct states (excluding the clock). Chess also has a discrete set of percepts and actions. Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time. Taxi-driving actions are also continuous (steering angles, etc.). Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.Discrete

Continuous

KNOWN VS. UNKNOWN: Strictly speaking, this distinction refers not to the environment itself but to the agent’s (or designer’s) state of knowledge about the “laws of physics” of the environment. In a known environment, the outcomes (or outcome probabilities if the environment is

nondeterministic) for all actions are given. Obviously, if the environment is unknown, the agent will have to learn how it works in order to make good decisions.

Known

Unknown

The distinction between known and unknown environments is not the same as the one between fully and partially observable environments. It is quite possible for a known environment to be partially observable—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over. Conversely, an unknown environment can be fully observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them.

As noted on page 39, the performance measure itself may be unknown, either because the designer is not sure how to write it down correctly or because the ultimate user—whose preferences matter—is not known. For example, a taxi driver usually won't know whether a new passenger prefers a leisurely or speedy journey, a cautious or aggressive driving style. A virtual personal assistant starts out knowing nothing about the personal preferences of itsnew owner. In such cases, the agent may learn more about the performance measure based on further interactions with the designer or user. This, in turn, suggests that the task environment is necessarily viewed as a multiagent environment.

The hardest case is partially observable, multiagent, nondeterministic, sequential, dynamic, continuous, and unknown. Taxi driving is hard in all these senses, except that the driver's environment is mostly known. Driving a rented car in a new country with unfamiliar geography, different traffic laws, and nervous passengers is a lot more exciting.

Figure 2.6 lists the properties of a number of familiar environments. Note that the properties are not always cut and dried. For example, we have listed the medical-diagnosis task as single-agent because the disease process in a patient is not profitably modeled as an agent; but a medical-diagnosis system might also have to deal with recalcitrant patients and skeptical staff, so the environment could have a multiagent aspect. Furthermore, medical diagnosis is episodic if one conceives of the task as selecting a diagnosis given a list of symptoms; the problem is sequential if the task can include proposing a series of tests, evaluating progress over the course of treatment, handling multiple patients, and so on.

Figure 2.6

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Examples of task environments and their characteristics.

Figure 2.6 Examples of task environments and their characteristics. We have not included a "known/unknown" column because, as explained earlier, this is not strictly a property of the environment. For some environments, such as chess and poker, it is quite easy to supply the agent with full knowledge of the rules, but it is nonetheless interesting to consider how an agent might learn to play these games without such knowledge. The code repository associated with this book (aima.cs.berkeley.edu) includes multiple environment implementations, together with a general-purpose environment simulator for evaluating an agent's performance. Experiments are often carried out not for a single environment but for many environments drawn from an environment class. For example, to evaluate a taxi driver in simulated traffic, we would want to run many simulations with different traffic, lighting, and weather conditions. We are then interested in the agent's average performance over the environment class.